「堅く柔らかく…数理計画アプローチ再訪特集号」 解　説

ここまで解ける整数計画

宮代 隆平* 松井 知己†

1. はじめに

近年の計算機パワーの増大により，様々な分野において以前には計算不可能であった大規模な問題が扱えるようになっている．さらに数理計画の世界では，最適化アルゴリズムそのものがハードウェアの進歩に勝るとも劣らない速度で進化しており，最先端のアルゴリズムを実装した最適化ソルバーの性能は数年前に比べて飛躍的に向上している．本稿では整数計画問題を取り上げ，ベンチマークを通して現在の最適化ソルバーの性能を紹介する．また，整数計画モデルがうまく解けない場合のガイドを簡単に記す.「整数計画を使おうと思うが，どのくらいの規模の問題まで解けるのかが知りたい」「以前に整数計画問題としてモデル化したが，計算時間がかかりすぎた」という方に，本稿が参考になれば幸いである．

2. 整数計画モデル

整数計画問題は，広い意味では「与えられた制約式および（一部に）整数条件がついた変数の下で，目的関数の値を最適化する問題」であるが，通常は「与えられた線形制約式および（一部に）整数条件がついた変数の下で，線形目的関数の値を最小化／最大化する問題」を考える．例えば，全ての変数が 0 または 1 を取る 0-1 整数計画問題は以下の (1) 式で表される．

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{c}^\top \boldsymbol{x} \\ &\text{subject to} \quad A\boldsymbol{x} \geq \boldsymbol{b}, \\ &\qquad\qquad \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{1}$$

なお本稿では誌面の都合上，整数計画法そのものの詳細な解説は行わない．整数計画法については [14,21,24], 線形計画法については [15] を参照のこと．

さて，いま手元に解かなければいけない最適化問題があるとする．もちろんこの問題が解析的に解ければうれしいが，通常扱う問題は NP-困難なことが多いはずだ．これに対しては，発見的解法を実装するか，整数計画問題としてモデル化しソルバーに解かせるなどのアプローチがある．整数計画法を利用するアプローチには，発見的解法を用いる場合と比較して以下のメリットが考えられる．

- 最適性の証明が得られる

分枝限定法により，最適性の証明が（列挙の果てに）行える．たとえば公平性を重視しなければいけない問題の場合，解が最適であることが非常に重要になる．

- 下界／上界が出る

実用的には,「これ以下／以上の目的関数値を持つ許容解は存在しない」という情報だけでも十分な場合がある．分枝限定法は線形緩和問題を利用しているので，計算が終了しなくてもその時点での下界／上界の情報を得ることができる．

- 問題が不能の場合

発見的解法を用いていて許容解が見つからない場合，まずはパラメータの調整・アルゴリズムの改良などを行うのが普通だが，問題が不能だった場合には時間の浪費になってしまう．これに対して分枝限定法では,「問題が不能である」ということを（時間をかければ）確認できる．

- プログラム開発が不要

整数計画問題それ自体はソルバーに解かせることにすれば，自前でプログラムを開発する必要が無く，モデル化に専念すればよいためトータルの時間が大幅に節約できる．また，実際に解く問題が「最小化／最大化」ではなく,「条件を全て満たす答えがあるか？」という形になっている場合も多いが，整数計画ソルバーをこの種の問題に対する探索プログラムとして用いることももちろん可能である．

それでは，解きたい問題を整数計画問題としてモデル化することにしよう．現在，ほぼ全ての整数計画ソルバーは，緩和問題として線形計画問題を利用している．したがって，整数計画モデルを立てる際には，線形制約式・線形目的関数でなくてはならない．こう書くとかなり限定されているようだが，一見異なるクラスの問題も，変数の追加・式の変形により整数計画モデルとして定式化できる．

例えば，工業製品の誤差評価などの場面などにおいて，複数の目的関数 $\boldsymbol{c}_1^\top \boldsymbol{x}, \boldsymbol{c}_2^\top \boldsymbol{x}, \ldots, \boldsymbol{c}_k^\top \boldsymbol{x}$ があるとき，こ

* 東京農工大学 大学院共生科学技術研究院
† 中央大学 理工学部情報工学科
Key Words: integer programming, software, benchmark

れらの最大値を最小化したい場合がある．このような min max 形の目的関数は，新しい変数 z および制約式 $\boldsymbol{c}_1^\top \boldsymbol{x} \le z, \boldsymbol{c}_2^\top \boldsymbol{x} \le z, \ldots, \boldsymbol{c}_k^\top \boldsymbol{x} \le z$ を追加して，目的関数 z を最小化することにより，線形不等式系で表すことができる．同様にして，最小値の最大化も線形不等式系で表現できる．また，絶対値（の和）の最小化を目的関数に入れることができる．例えば $|x_1|+|x_2|+\cdots+|x_n|$ を最小化したい場合は，変数 $z_1, z_2, \ldots, z_n$ を導入し，$-z_i \le x_i \le z_i\,(i=1,2,\ldots,n)$ という制約のもとで，$z_1+z_2+\cdots+z_n$ を最小化すれば良い．残念ながら，絶対値の和の最大化は定式化が困難である．以上は，整数変数を含まない線形計画問題についても適用できる．

整数変数は，離散的な値を取る事象のモデル化に使用する他に，その離散性を利用して定式化のトリックに使うことができる．例えば，0-1 変数は「制約式のスイッチ」の役割として用いることができる．非負の連続変数 $x_1, x_2, \ldots, x_n$ を，制約条件 $A\boldsymbol{x} \ge \boldsymbol{b}$ の下で，制約式 $x_i \le d_i$ $(i=1,2,\ldots,n)$ に違反する個数を最小化したい，という問題を考えよう．このような問題は，0-1 変数 $y_1, y_2, \ldots, y_n$ および十分に大きな定数 M を導入することにより以下の (2) 式として定式化できる．

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} y_i \\ \text{subject to} \quad & A\boldsymbol{x} \ge \boldsymbol{b}, \\ & \boldsymbol{x} - \boldsymbol{d} \le M\boldsymbol{y}, \\ & \boldsymbol{x} \ge \boldsymbol{0}, \\ & \boldsymbol{y} \in \{0,1\}^n. \end{aligned} \tag{2}$$

このように，大きな定数を用いる定式化の手法を big-M 法と呼ぶ．big-M 法を用いると，さまざまな論理関係が定式化できるが，M を選択する際にあまりに大きな値を用いると数値計算の面で不安定になるので，定数 M は制約式の意味を変えない範囲でなるべく小さいものを選択するのが良い．

その他，非線形制約式なども人工的な変数を導入すると線形制約式に書きなおせる場合があり，様々なクラスの問題を整数計画モデルとして表すことが可能である．これら定式化の様々なテクニックについては [17,20] などを参照してほしい．

さて，解くべき問題を整数計画問題としてモデル化したのはよいが，はたして現実的な時間内に答えを求められるのだろうか．定式化の結果として数千変数，数万変数の問題になってしまった場合，10 年前までは実質的に別のアプローチを考えざるを得なかった．しかし現在の最適化ソルバーは，以前と比較にならないほど巨大な問題を解くことができ，toy problem だけでなく実務上の最適化問題が次々に解決されている．たとえば分枝限定法は，10 年前と比較すると**ハードウェアの進歩を除いたうえで** 1000 倍以上高速になっているが，これは前処理，分枝戦略，カットなどの各種アルゴリズムの進歩，およびベースとなる単体法の高速化によってもたらされたものである．(線形計画法・整数計画法の発展の歴史については [16,2–4] が詳しい.）次節では，ベンチマーク問題を通して整数計画ソルバーが現時点でどのくらいの規模の問題を扱えるのか紹介する．

3. 整数計画問題のベンチマーク

ここでは整数計画問題のベンチマークとして，ベンチマークセット MIPLIB 2003[1] を最新の整数計画ソルバーで解いた結果を記す．今回の実験では，MIPLIB 2003 に用意されている 60 問のうち,「商用ソフトで 1 時間以内に解けるクラス」に分類されていた 28 問について実験を行った．

計算には，Windows XP (CPU: Pentium 4, 2.8GHz, RAM: 1024MB) 上の整数計画ソルバー ILOG CPLEX 10.0[13] (2006 年 1 月リリース) を使用した．なお比較のために，ILOG CPLEX 9.0/8.1/7.0 (それぞれ 2003 年 12 月／2002 年 12 月／2000 年 10 月リリース) も使用した．CPLEX のパラメータ設定は，基本的にデフォルトのまま計算させた[1].

第 1 表に，ベンチマーク問題の規模および計算結果を示す．各列は左からそれぞれ問題名，制約式の本数，変数の個数，そのうち整数変数の個数，行列 A に含まれる非零要素の数，CPLEX 10.0/9.0/8.1/7.0 での計算時間（単位 : 秒）を示している．数千変数／制約式の問題も実用的な時間で解けており，また概して難しい問題に対しては大幅なスピードアップが見られ，最適化アルゴリズムの進化が窺える.（一部の問題では，新しいソルバーの方が遅くなっているが，これは分枝順序などの差異によるものである.）現在もアルゴリズムの改良は続いており，古いベンチマーク問題はどんどんベンチマークとしての用を成さなくなっている．

なお第 1 表に示した計算時間は,「最適解が求まり，なおかつその解が最適であると証明できた」ところまでの計算時間である．実用上では,「最適かどうかはともかく質の良い解を求めたい」というケースが多く，この場合はさらに大規模な問題まで扱える．第 2 表は，第 1 表の CPLEX 10.0 で 100 秒以上かかったベンチマーク問題について，最適解が見つかった時点までの計算時間を示したものである．最近のソルバーは，質の良い解を高速に見つける能力が非常に高くなっており，最適解は計算の早い過程で見つかり，その最適性を証明するのに計算時間の大部分を要するという状況が増えてきているが，第 2 表からもそれがわかる．

また，各種ソルバーの設定パラメータ（カット生成，分枝戦略など）を調整することにより，問題がより高速

[1] ベンチマーク問題 `manna81`, `opt1217` に対してのみ，Gomory cut と mixed integer rounding cut を多数生成する設定にした．

第 1 表　ベンチマーク問題の規模と計算時間（計算時間の単位：秒）

問題名	制約式	変数	整数変数	非零要素	CPLEX 10.0	CPLEX 9.0	CPLEX 8.1	CPLEX 7.0
10teams	230	2025	1800	12150	29.2	7.8	4.9	6.4
aflow30a	479	842	421	2091	39.6	81.5	191.2	142.2
air04	823	8904	8904	72965	25.5	27.3	44.0	72.4
air05	426	7195	7195	52121	23.5	29.1	31.8	57.6
cap6000	2176	6000	6000	48243	0.9	0.5	2.0	6.4
disctom	399	10000	10000	30000	58.0	777.7	1044.3	8989.9
fiber	363	1298	1254	2944	0.5	0.3	0.4	0.6
fixnet6	478	878	378	1756	1.3	0.4	0.3	0.3
gesa2	1392	1224	408	5064	0.5	0.5	1.3	0.9
gesa2-o	1248	1224	720	3672	4.3	3.9	1.4	1.2
manna81	6480	3321	3321	12960	0.6	0.6	0.5	0.5
mas74	13	151	150	1706	1380.5	1699.4	1855.1	1424.7
mas76	12	151	150	1640	154.4	85.2	110.3	92.8
misc07	212	260	259	8619	17.4	78.2	97.9	87.9
mod011	4480	10958	96	22254	92.7	101.2	983.3	failure
modglob	291	422	98	968	0.2	0.5	0.5	0.4
mzzv11	9499	10240	10240	134603	495.6	787.9	10000<	10000<
mzzv42z	10460	11717	11717	151261	97.5	90.9	10000<	10000<
nw04	36	87482	87482	636666	40.9	28.0	23.3	8.1
opt1217	64	769	768	1542	0.4	5.6	10000<	10000<
p2756	755	2756	2756	8937	0.5	0.6	0.5	0.6
pk1	45	86	55	915	120.9	106.5	203.0	102.9
pp08a	136	240	64	480	1.2	1.6	1.3	1.4
pp08aCUTS	246	240	64	839	3.3	2.4	4.5	2.7
qiu	1192	840	48	3432	64.4	209.9	120.9	728.8
rout	291	556	315	2431	127.8	2800.6	3412.1	673.2
set1ch	492	712	240	1412	0.7	0.8	0.9	0.6
vpm2	234	378	168	917	1.2	0.8	1.0	6.4

第 2 表　最適解発見までの時間（単位：秒）

問題名	総計算時間	最適解の発見
mas74	1380.5	9.9
mzzv11	495.6	434.1
mas76	154.4	0.6
rout	127.8	90.9
pk1	120.9	12.5

に解けることもある．どのパラメータの調整が有効かは，問題構造に対する深い洞察が必要なことが多く，試行錯誤になってしまう場合も多々あるが，求解速度が大きく変わる場合もあるので試してみる価値はある [2,5].

では実用的には，ソルバーはどの程度の問題まで解けるのだろうか？今回扱ったベンチマーク問題では，変数の個数・制約式の本数ともに 1 万を超えているベンチマーク問題 mzzv42z も 100 秒未満で解けている．ただし第 1 表からもわかるように，問題の難しさは単純に変数の個数や制約式の本数で決まるわけではなく，サイズが小さくても難しい問題[1]もある [6]．問題の難易度を決めるのは，行列 A の形や変数の形式（0-1 変数か，一般の整数変数か，または連続変数も含むか），さらに「線形緩和問題が元の整数計画問題にどのくらい “近いか”」である．同じ最適化問題を表している整数計画モデルでも，上記のような定式化の良し悪しにより解けるスピードは劇的に変化する．残念ながら（問題が本質的に難しい場合や）モデル化がよくない場合には，解ける問題のサイズはまだそれほど大きくなく，数百変数くらいで行き詰まってしまうことも珍しくはない．次節では，定式化した問題がうまく解けない場合のガイダンスを示す．

市販されている最適化ソルバーは数多くの種類があるが，数理計画の分野では商用ソフト CPLEX[13], XPRESS-MP[8] などがよく用いられている．国産では NUOPT[23] などのソフトウェアがある．商用以外のソ

[1] 例えば，MIPLIB 2003[1] の timtab2 というベンチマーク問題は制約式が 294 本，変数が 675 個の問題であるが，未だに最適解がわかっていない．

フトウェアも，GLPK[11,12], lp_solve[19] ほか多数あるが，整数計画法に関しては商用ソフトウェアの方がまだ圧倒的に高性能である．ソフトウェアの機能についてのサーベイは, [5,10,18] などが参考になる．なお，最近の整数計画ソルバーでは線形制約・線形目的関数以外の整数計画問題（例えば凸 2 次制約／凸 2 次目的関数）を扱えるものもあるが，当然ながらこれらはより難しい問題となり，大きなサイズの問題を実用的な時間で解くのはまだ困難である．しかし，最適化アルゴリズムの進化が早いため，ソルバーが扱える問題サイズがさらに拡大していくことは間違いないだろう．

4. 早分かりガイド

整数計画問題を最適化ソルバーで解いてみたとき，思うような性能が得られなかったらどうするのか？例えば，本当に解きたい問題の前に小さなモデルでテストしたところ，商用の最適化ソルバーで数時間かかってしまった場合などである．本節ではそんなときの典型的な手続きを，個人的な経験と勘に基づいて簡単に記してみる．以下の記述は，最適化ソルバーの急速な進歩からすると，数年後には見当違いの事となっている可能性もあるので注意されたい．

問題が思うようなスピードで解けないときの解決法は，大きく分けて以下の 2 つがある．

(1) あきらめる

あきらめが肝心なときもある．ここで言う「あきらめる」とは，

- 新たに仮定等を導入してモデル自体を改訂する
- 元の問題をより小さな問題に分割する
- 発見的解法を作ることにする
- とりあえず計算機を走らせておいて，
  整数計画法に詳しい人を探しにいく

などがある．

(2) あきらめない

もちろん簡単にあきらめてはいけない．そんなときは第 3 表の手順で状況の改善を図るのが良いだろう．ただし以下では，0-1 整数計画で非負関数の最小化を目的に持つ問題を扱っているとする．

第 3 表 基本的なチェック項目

(a) 線形計画緩和問題の最適解を見る
(b) 最適解発見までの時間を見る
(c) 緩和問題の性能を見る

(a) 線形計画緩和問題の最適解を見る

ほとんどの最適化ソルバーでは，線形緩和問題を内部で解いている．近年は線形計画問題は非常に高速に解けるので，通常は問題ないはずだ．ここで注目するのは，線形緩和問題の最適解において 0 または 1 となっている変数の個数である．もし全ての変数が 0 または 1 になっていれば，もちろんこれは元の 0-1 整数計画の最適解となっている．さて，0-1 変数全体のうち，0 または 1 となっている変数が，変数全体に対し何%くらいを占めているかを見てみよう．

第 4 表 ケース別ガイド

	最適解発見 早い	最適解発見 遅い
緩和問題の性能が良い	Case 1 多最適解の解消	Case 2 許容領域の拡張
緩和問題の性能が悪い	Case 3 制約の追加 変数の一部丸め	Case 4 許容領域の拡張 あきらめる？

(a-1) 0 または 1 となっている変数の個数が 10%以下だ．

このようなときは，問題自体が非常に難しいことを覚悟したほうが良い．以下に記すいくつかの方法を少しためしてみて，顕著な改善が見られないならば，適当なところで見限って「(1) あきらめる」に戻った方が良い．モデルの改訂や問題の分割を行う際は，0 または 1 となっている変数の個数が増加するように行うのが得策だが，具体的な方法は個々の問題に大きく依存する．

(a-2) 0 または 1 となっている変数の個数が 90%以上だ．

このような場合，通常は速く解けることが多いのだが，なぜか時間がかかっているというのは，何か難しい構造が一部に隠れていることが多い．整数になっていない変数からなる問題を想定して，それがどんな問題なのか改めて考える必要があるだろう．例えば，巡回セールスマン問題，最大カット問題，p-センター問題など，典型的な難しい問題（に良く似た構造）が隠れていないか確認し，それぞれの解法において成功している制約式を導入する必要がある．ちなみに，線形緩和問題の最適解で 0 または 1 になっている変数の値は，元の整数計画問題の最適解で同じ値を持つとは限らないことに注意されたい．

(a-3) それ以外のときは，以下に進もう．

(b) 最適解発見までの時間を見る

最適解発見までの時間が，総計算時間の何%になっているかを確認する．25%を超えていたら，遅い方と思った方が良い．

(c) 緩和問題の性能を見る

線形緩和問題の最適値が，元の整数計画問題の何%くらいになっているかを確認する．70%以下だったら，線形緩和問題の性能が悪いと思った方が良い．

上記の 2 つのデータから，4 通りの場合（第 4 表参照）があるが，これらの典型的な状況を挙げよう．

Case 1: 最適解発見は早い，線形緩和問題の性能は良い．

これなら計算時間がもっと短くてもよいのに？と思うケースである．このような事態は，最適解が異常にたく

さんある時によく起こる．その理由と対策は以下のようなものである．

(i) 目的関数が無い：

「許容解を求めれば良い問題だから」といって目的関数を作っていないと，総計算時間の増加を確実に招く．あっても無くても良いなら，目的関数は必ず入れよう．何でもいいなら，係数は整数乱数を使うのが良い．

(ii) 目的関数の係数がすべて 1 である：

よく誤解されるが，目的関数の係数がばらばらな値の方が一般に速く解ける．何か意味をつけて目的関数の係数をいろんな値にし，最適解の個数を絞り込むのが良い．論文等の計算実験では，人工的な問題として目的関数の係数を (整数) 乱数で生成している場合も多いが，これを 0 または 1 にすると突然総計算時間が極端に増加することは珍しくない．「すべて 1」がどうしても必要なら，乱数を使って係数を摂動するという方法がある．摂動した際は，目的関数の係数を適当にスケーリングして全て整数にしておいた方が良いこともある．

(iii) 目的関数が min max 形である：

前述した min max 形の目的関数は，実は総計算時間の増加を招くのでなるべくなら扱わないほうが良い．典型的な例が min max 形の目的関数を持つ p-センター問題であろう．良く似た設定で，目的関数が単純な p-メディアン問題の方がずっと容易に解けることが知られている．複数の目的関数値 $\boldsymbol{c}_1^\top\boldsymbol{x}, \boldsymbol{c}_2^\top\boldsymbol{x}, \ldots, \boldsymbol{c}_k^\top\boldsymbol{x}$ を均等にしたいなら，最大値の最小化ではなく，絶対値の和 $|\boldsymbol{c}_1^\top\boldsymbol{x}|+|\boldsymbol{c}_2^\top\boldsymbol{x}|+\cdots+|\boldsymbol{c}_k^\top\boldsymbol{x}|$ の最小化を目的関数にすることを勧める．

Case 2: 最適解発見は遅い，線形緩和問題の性能は良い．

このような状況は，許容解が非常に少なく，問題が不能になりやすいとき起こることが多い．本質的にはモデルを見直す必要があるが，以下のような方法がうまくいく場合もある．まず制約式を見直し，必ず満たさねばならない制約 (hard constraints) と，できれば満たして欲しい制約 (soft constraints) に分ける．その上で「できれば満たして欲しい制約」を緩和する．モデルを変更して良いなら，制約が等式 $\boldsymbol{a}^\top\boldsymbol{x}=b$ ならその代わりに制約 $b-\varepsilon\le\boldsymbol{a}^\top\boldsymbol{x}\le b+\varepsilon$ とするのも良いだろう（ただし ε は適当な小さな正数である）．新しい変数 z と罰金係数 θ を導入し，例えば制約が等式 $\boldsymbol{a}^\top\boldsymbol{x}=b$ ならば，その代わりに制約 $b-z\le\boldsymbol{a}^\top\boldsymbol{x}\le b+z$ を入れ，目的関数に新たな項 $+\theta z$ を加えるという方法もある．(制約が不等式 $\boldsymbol{a}^\top\boldsymbol{x}\ge b$ ならば，その代わりに制約 $\boldsymbol{a}^\top\boldsymbol{x}+z\ge b$ を入れ，目的関数に新たな項 $+\theta z$ を加える．)

Case 3: 最適解発見は早い，線形緩和問題の性能は悪い．

線形緩和の性能を上げるために，何か新しい不等式制約を見つけて加える必要がある．もし最適性を犠牲にして良いのなら，緩和問題の最適解で 0 または 1（に近い値）をとっている変数を，それぞれ 0 または 1 に丸めて，問題のサイズを縮小して最適化ソルバーで解いてみるのも良い．あるいは単に切り上げるのではなく，ランダム丸め法を繰り返し適用するのも良い．これは，緩和問題での最適値を確率だと思って（いくつかの変数を）0 または 1 に丸め，(残った変数からなる）得られた問題を最適化ソルバーにかけるというものである．(例えば変数 x_1 が，線形緩和問題の最適解で 0.8 という値になっていたら，0.8 の確率で x_1 の値を 1 に固定し，0.2 の確率で x_1 の値を 0 に固定する．）許容解が少ない場合は，このような丸め法は，変数を固定してできる問題が不能となってしまうことが多いため残念ながらうまく働かない．

また変数の間に対称性がある場合，線形緩和問題の性能はかなり悪くなる．このような時は $x_1\le x_2\le\cdots\le x_n$ のように変数に無理やり区別をつけてやり，対称性を除去するべきである．どの変数の間に対称性があるかの判断は一般には難しいため，数式を見るよりモデル化の時点から無くすように注意するのが得策である．

Case 4: 最適解発見は遅い，線形緩和問題の性能は悪い．

非常に状況が悪いため「(1) あきらめる」に戻ったほうが良いかもしれない．とりあえず，以下の事項を見直してみよう．

- 許容解は本当に使えるか？

実際に現実問題のモデル化を請け負って，最適解を求めると「こんな答えでは使えない」となることがしばしばある．これは，モデル化の際の「制約条件の洗い出し」が不十分なために起こる．時間をかけて最適解を求める前に，許容解を見て制約式が不足していないか確認した方がよい．制約式を追加すると，計算時間が劇的に変化することはしばしばある．

- 問題から得られる知見は全て使っているか？

モデル化の際に，「この制約式は，これまでに作った制約式を満たしていれば自動的に満たしているはず」というような場合でも，とりあえず追加しておくことを奨める．これは，整数条件を含んだ世界で許容領域が同じでも，線形緩和した場合には許容領域が異なることがあるからである．また，ソルバーによるカットの生成を促す意味もある．冗長な制約式はたいていの場合問題にならないので，思いついた制約式は全て入れるくらいのつもりでもよい．

また，問題に対して持っている先験的な知識（暫定解，上界／下界値ほか）は無いだろうか？例えば，ある変数群を固定すると問題が著しく縮小されるような場合，それに含まれる変数から分枝する分枝順序を導入してみるとよい．最近では許容解から別の許容解を見つける手法も強力になっているため（[7,9] など），あらかじめ知っている許容解を与えてやると良いこともある．

5. おわりに

数理計画の分野では，整数計画法はもちろん重要な研究対象の一つであるが，それ以外の研究分野においても，整数計画を用いた最適化が加速度的に普及している．例えば，四色定理に関連したグラフ理論の問題が，整数計画法を用いて解かれている [22]．また，工場の生産現場などでは従来から最適化が行われてきたが，顧客からの受注に合わせて即時に生産計画を再最適化するといった「オンライン最適化」も，既にたくさんの実施例が報告されている．これからはどのような分野においても，最適化問題に対する整数計画のモデル化は，必ず試してみるべき手段の一つになっていくのではないだろうか．

（2006 年 4 月 12 日受付）

参 考 文 献

[1] T. Achterberg, T. Koch, A. Martin: MIPLIB 2003; *Operations Research Letters*, Vol. 34, No. 4, pp. 361–372 (2006) http://miplib.zip.de/

[2] R. E. Bixby: MIP: Theory and practice — closing the gap; Conference and Research Papers, ILOG (2000) http://www.ilog.com/products/optimization/tech/research/mip.pdf

[3] R. E. Bixby: Solving real-world linear programs: a decade and more of progress; *Operations Research*, Vol. 50, No. 1, pp. 3–15 (2002)

[4] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling: Mixed-integer programming: a progress report, *The Sharpest Cut* (M. Grötschel ed.), SIAM, pp. 309–327 (2004)

[5] A. Atamtürk, M. W. P. Savelsbergh: Integer-programming software systems; *Annals of Operations Research*, Vol. 140, No. 1, pp. 67–124 (2005)

[6] G. Cornuéjols, M. Dewande: A class of hard small 0-1 programs, *Integer Programming and Combinatorial Optimization: 6th International IPCO Conference*, R. E. Bixby, E. A. Boyd, R. Z. Rios-Mercado (Eds.), Lecture Notes in Computer Science, Vol. 1412, Springer, pp. 284–293 (1998)

[7] E. Danna, E. Rothberg, C. Le Pape: Exploring relaxation induced neighborhoods to improve MIP solutions; *Mathematical Programming*, Series A, Vol. 102, No. 1, pp. 71–90 (2005)

[8] Dash Optimization: Xpress-MP; http://www.dashoptimization.com/

[9] M. Fischetti, A. Lodi: Local branching; *Mathematical Programming*, Series B, Vol. 98, Nos. 1–3, pp. 23–47 (2003)

[10] R. Fourer: Linear programming — software survey; *OR/MS Today*, Vol. 32, No. 3, pp. 46–55 (2005) http://www.lionhrtpub.com/orms/orms-6-05/frsurvey.html http://www.lionhrtpub.com/orms/surveys/LP/LP-survey.html

[11] GLPK (GNU linear programming kit); http://www.gnu.org/software/glpk/glpk.html

[12] Glpk for windows; http://gnuwin32.sourceforge.net/packages/glpk.htm

[13] ILOG: CPLEX; http://www.ilog.co.jp/

[14] 今野浩，鈴木久敏：整数計画法と組合せ最適化；日科技連 (1982)

[15] 今野浩：線形計画法；日科技連 (1987)

[16] 今野浩：役に立つ一次式 — 整数計画法「気まぐれな王女」の 50 年；日本評論社 (2005)

[17] 久保幹雄，田村明久，松井知己（編集）：応用数理計画ハンドブック；朝倉書店 (2002)

[18] J. T. Linderoth, T. K. Ralphs: Noncommercial Software for Mixed-Integer Linear Programming, *Integer Programming: Theory and Practice* (J. Karlof ed.), CRC Press, pp. 253–303 (2005)

[19] Lp_solve; http://groups.yahoo.com/group/lp_solve/

[20] 森雅夫，松井知己：オペレーションズ・リサーチ（経営システム工学ライブラリー 8）；朝倉書店 (2004)

[21] G. Nemhauser, L. Wolsey: Integer and Combinatorial Optimization; Wiley-Interscience (1999)

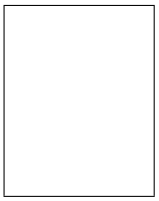
[22] G. Pataki, S. Schmieta, S. Ceria, M. Ferris, J. Linderoth: Seymour is solved!; http://www-unix.mcs.anl.gov/metaneos/seymour/

[23] 数理システム：NUOPT; http://www.msi.co.jp/nuopt/

[24] L. Wolsey: Integer Programming; Wiley-Interscience (1998)

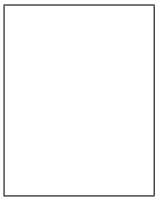
著 者 略 歴

宮代 隆平 (非会員)



2004 年 3 月東京大学大学院情報理工学系研究科数理情報工学専攻博士課程修了．同年 6 月東京農工大学大学院共生科学技術研究部助手となり現在に至る．組合せ最適化の研究に従事．博士（情報理工学）．オペレーションズ・リサーチ学会などの会員．

松井 知己 (非会員)



1990 年 3 月東京工業大学大学院 総合理工学研究科システム科学専攻博士後期課程修了．同年 4 月東京理科大学理工学部経営工学科助手．1992 年 4 月東京大学工学部計数工学科講師．1996 年 4 月東京大学大学院工学系研究科計数工学専攻助教授．2006 年 4 月中央大学理工学部情報工学科教授となり現在に至る．組合せ最適化の研究に従事．博士（理学）．オペレーションズ・リサーチ学会などの会員．